Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

WUCSE-2004-65

Towards a Perception Based Image Editing System

Authors: Bailey, Reynold; Bujans, Raquel; Grimm, Cindy

October 19, 2004

Towards a Perception Based Image Editing System

Reynold Bailey*, Raquel Bujans†, and Cindy Grimm‡
Washington University in St. Louis

1 Introduction

The primary goal of this research is to develop a perception based image editing system. The input to this system will be either a rendered image, a photograph, or a high dynamic range image. We are currently developing techniques that allow the user to edit these images in a perceptually intuitive manner. Specifically we are considering the following image editing features:

1. Warm-cool image adjustment.
2. Intensity adjustment.
3. Contrast adjustment.
4. Detail adjustment.

The algorithms we are developing can be used either in an interactive editing system or for automatic image adjustment.

2 Overview

Analysis of the chromatic properties of neurons in the visual system of the macaque monkey led to the development of the DKL color model [1]. The visual system of the macaque is closely related to that of humans. As such, we have decided to use the DKL color model in our research. In Section 3 we introduce the DKL color model and present several color space conversions involving the DKL color model. One of the goals of our work is to take a given image and make parts of it look more warm or cool. Using this approach, an artist could take a picture of a sunrise and "paint" it to look more like a sunset. We have developed a technique to generate tables of color

*email:rjb1@cse.wustl.edu
†email:rb3@cec.wustl.edu
‡email:cmg@cse.wustl.edu

shifts. These tables show how a given hue looks when it is made to be “more warm” or “more cool”. This work is presented in Section 4. Another of our goals is to develop techniques to adjust the intensity of some input image. Our research in this area was presented as a poster at SIGGRAPH 2004 [2] and is discussed in Section 5. We discuss future directions of our work in Section 6.

3 DKL Color Model

In 1984, Derrington et al. [1] presented research in which they analysed the chromatic properties of neurons in the visual system of the macaque. In their work, they developed a color coding system which later became known as the DKL color model. Incorporated into the DKL color model is a mechanism for simulating the response of the eye’s cones and double opponent cells. The cones are cells that sense color. They only trigger a response if enough light is present - a certain amount of light must be reflected off the objects in order for them to be perceived as being a certain color. There are three types of cones that are sensitive to short, medium, and long wavelengths of visible light. The double opponent cells, on the other hand, react when opposing colors are placed next to each other. There are two main types of double opponent cells - a red-green type and a blue-yellow type. The red-green type fires a response when the eyes see a red color surrounded by a green color, or when the eyes see a green color surrounded by a red color. The blue-yellow pair behaves in the same manner. The double-opponent cell response is what causes us to perceive that placing a bright red and a bright green right next to each other makes them both seem even more bright, almost to the point of hurting our eyes. The double-opponent response is separated into its own unique channels in DKL. Much like how RGB is separated into 3 channels representing red, green, and blue, DKL is separated into 3 channels representing the red-green, blue-yellow, and intensity responses. [1]

3.1 Conversion from RGB to DKL

The phosphor luminance response curves for R,G,and B vary from monitor to monitor, but in general follow the same trend. Using data provided to us from Phosphor Technology Ltd., we were able to plot the response curves of the R, G, and B phosphors in a typical monitor. We also obtained experimental data from the Colour and Vision Research Laboratories, University of California, San Diego that allow us to plot the response curves of the short(S), medium(M), and long(L) wavelength sensitive cones.

The conversion from RGB to DKL proceeds in two smaller steps: conversion from RGB to SML, and conversion from SML to DKL. Both conversions are least squares problems of the form $Ax = b$.

[1] The DKL model can also take into account the response from the rods in our eyes, however that has not yet been incorporated into our work.

3.1.1 RGB to SML

Converting from RGB to SML means taking a user defined color C_u and expressing it in terms of the short, medium and long curves. As a preprocessing step we first take the phosphor response curves F for R, G, and B and dot product them with the S, M, and L response curves to obtain new scaled phosphor curves:

$$F_{SR} =< F(R), S > \qquad F_{MR} =< F(R), M > \qquad F_{LR} =< F(R), L >$$

$$F_{SG} =< F(G), S > \qquad F_{MG} =< F(G), M > \qquad F_{LG} =< F(G), L >$$

$$F_{SB} =< F(B), S > \qquad F_{MB} =< F(B), M > \qquad F_{LB} =< F(B), L >$$

The scaled phosphor curves are then multiplied by C_u (made of a triplet r, g, b) to form scaled S,M, and L curves:

$$F_{SRr} = F_{SR} * r \qquad F_{MRr} = F_{MR} * r \qquad FLRr = F_{LR} * r$$

$$F_{SGg} = F_{SG} * g \qquad F_{MGg} = F_{MG} * g \qquad FLGg = F_{LG} * g$$

$$F_{SBb} = F_{SB} * b \qquad F_{MBb} = F_{MB} * b \qquad FLBb = F_{LB} * b$$

All the S components of the scaled phosphor curves are added together to form to new scaled S curve S'. M' and L' are calculated similarly:

$$S' = F_{SRr} + F_{SGg} + F_{SBb}$$

$$M' = F_{MRr} + F_{MGg} + F_{MBb}$$

$$L' = F_{LRr} + F_{LGg} + F_{LBb}$$

S', M', and L' are then normalized to white. The conversion from RGB to SML can also be expressed as a least squares equation of the form $Ax = B$. A and x can be written as follows:

$$A = \begin{pmatrix} F_{SR} & F_{SG} & F_{SB} \\ F_{MR} & F_{MG} & F_{MB} \\ F_{LR} & F_{LG} & F_{LB} \end{pmatrix} \qquad x = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

Solving for B gives us:

$$Ax = B = \begin{pmatrix} S' \\ M' \\ L' \end{pmatrix}$$

3.1.2 SML to DKL

The conversion from SML to DKL is based on the following three formulas [4]:

$$R - G = \frac{M - L + 1}{2} \tag{1}$$

$$B - Y = \frac{\frac{M+L}{2} - S + 1}{2} \tag{2}$$

$$I = \frac{S + M + L}{3} \tag{3}$$

The R-G value models the behavior of the red-green double-opponent cells in our eyes, the B-Y models the behavior of the blue-yellow double-opponent cells, and the I value represents the color's intensity level. The SML to DKL conversion can also be written in the form of $Ax = B$. Let A and x be:

$$A = \begin{pmatrix} 0 & 0.5 & -0.5 \\ -0.5 & 0.25 & 0.25 \\ 0.33 & 0.33 & 0.33 \end{pmatrix} \qquad x = \begin{pmatrix} S \\ M \\ L \end{pmatrix}$$

Solving for B gives us:

$$Ax = B = \begin{pmatrix} R - G \\ B - Y \\ I \end{pmatrix}$$

The coefficients for A are taken from the R-G, B-Y, and I equations above. The S coefficients are placed into the first column, M in the second, and L in the third.

3.2 Conversion from DKL to RGB

The original RGB values can be solved for using the least-squares technique. Just like the conversion from RGB to DKL, the conversion back takes place in two steps. Given a color in DKL format C_{DKL} we must first convert it back to SML terms. Let A and B be:

$$A = \begin{pmatrix} 0 & 0.5 & -0.5 \\ -0.5 & 0.25 & 0.25 \\ 0.33 & 0.33 & 0.33 \end{pmatrix} \qquad B = \begin{pmatrix} R - G \\ B - Y \\ I \end{pmatrix}$$

x will hold the SML values. The new SML values must be "unnormalized", or multiplied by whatever values they were normalized by in the conversion to DKL. Next we take those scaled S,M,and L values and use them as the B for the second step:

$$A = \begin{pmatrix} F_{SR} & F_{SG} & F_{SB} \\ F_{MR} & F_{MG} & F_{MB} \\ F_{LR} & F_{LG} & F_{LB} \end{pmatrix} \qquad B = \begin{pmatrix} S' \\ M' \\ L' \end{pmatrix}$$

Solving for x gives us the original r,g,b values. In summary, we provide methods for the following color space conversions:

- RGB to DKL
- DKL to RGB
- RGB to SML
- SML to RGB
- SML to DKL
- DKL to SML

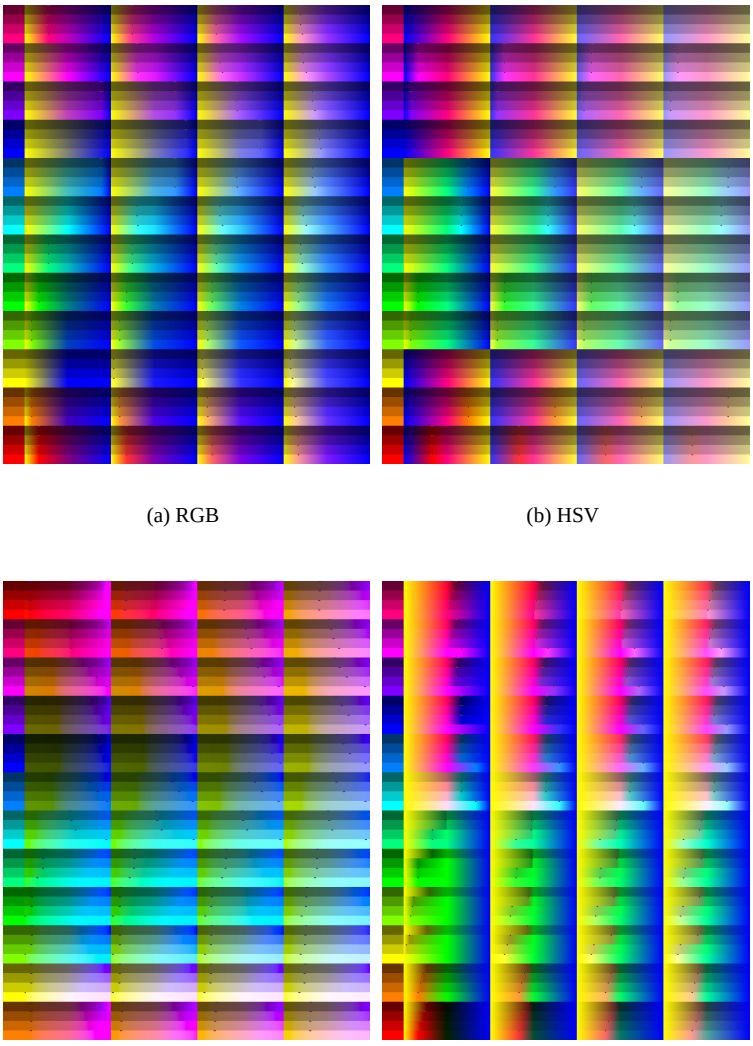
4 Color Shift Comparison Tables

We have developed a technique to generate tables of color shifts that show how a given hue looks when it is made to be "more warm" or "more cool". Tables also visually show us the limits of how much a given hue can be shifted before it looses it original color. Some colors can be shifted towards a given warm or cool color more than others. For example, green can be shifted further towards blue than red could. Using a table of shifts makes this information clear. We have generated color shift tables for the RGB, HSV, CIE LUV, and DKL color spaces, thereby facilitating the comparison of different colorspaces' shifting capabilities. These tables are shown in Figure 1.

We created these tables by first selecting 12 initial hues from around the color wheel shown in Figure 2(a). These colors have maximum value and saturation. The hues were placed into a vertical array. Next, we decremented each hue's value 3 times and added the resulting hues to the array. They were entered such that the original hue was followed by its 3 value variations. Finally, for each hue in our array (including the value variations), we decremented saturation level 3 times. We entered the saturation variations as a second dimension in our array. For each of the 48 positions in our vertical array, we entered 3 additional horizontal entries, corresponding to the saturation variations (Figure 2(b)). Let these initial hues be referred to as target hues T_h. Each hue in the 2-d array is then shifted towards a warm and a cool color. Let the warm color be denoted as C_w and the cool color be denoted as C_c. For the tables shown in Figure 1, we set C_w to yellow and C_c to blue. A gradient is generated for each that shows the shift from C_c, through our T_h, to C_w. While the rest of the implementation process differs from colorspace to colorspace, the process described here outlines the general idea used.

4.1 Gradient Generation

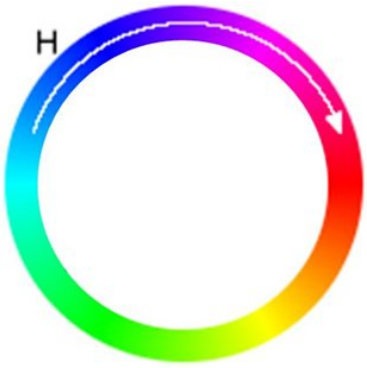
Each gradient in the table goes from C_w to C_c, passes through T_h, and a neutralized version of that T_h. The initial T_h's are often not neutral colors, but lean more towards C_w or C_c. A neutral version of the particular T_h is found and serves as that gradient's mid-point on the shift from C_w to C_c. It is equal parts C_w and C_c. Let the neutralized T_h be denoted as T_n. How the neutral is found depends on the colorspace and on C_w and C_c. For our purposes, we wanted a neutralized version of the T_h that was equal parts yellow and blue.



(a) RGB (b) HSV

(c) LUV (d) DKL

Figure 1: Color shift tables for the RGB, HSV, CIE LUV, and DKL color models.



H	S	S	S
V	S	S	S
V	.....		
V	.....		
H	.....		
V	.....		
V	.....		
V	.....		
	⋮		

(a) Color Wheel

(b) Color Shift Table Organization

Figure 2: Color wheel from which 12 initial hues are selected and a table showing how the color shift tables are organized.

Given that we know T_h should fall somewhere on the gradient between either C_w and T_n or T_n and C_c, we need to find where between the two it falls. In our tables, if it falls to the left of the neutral then we know our hue is more of a yellowish color. Otherwise, our hue is more bluish. In order to find T_h's desired location we use the following process. First we make an image of the gradient from the neutral to either the warm or cool hue, depending on which half our T_n is in. Keep in mind that the gradient's color varies horizontally (left to right), but not vertically. We do a three-way linear interpolation across the gradient in order to find the color that most closely resembles T_h. For each channel in RGB, we find the column in the gradient that is closest to our given T_n's channel, and then average the three resulting columns to find exactly where our T_n falls.

Lastly, a new gradient image is generated that incorporates T_n. A gradient from the closest C (either warm or cool) to T_h, T_h to T_n, and T_n to the farthest C is made. For example, if our T_h was red, then on a gradient between yellow and blue it would fall closer to red. Red as a T_h would fall somewhere on a gradient between yellow and its T_n. Therefore, we could create a final gradient that went from yellow to red, red to red's T_n, and red's T_n to blue. Once a final gradient is created for each original T_h, all the gradients are put together in table form.

4.2 Problems with DKL Table Generation

One problem with the DKL colorshift table is caused by the fact that not all colors in DKL map back to RGB. When we create neutralized versions of T_h's, the resulting T_n's do not always exist in RGB. As a temporary solution, all values converted back from DKL are clipped from 0 to 1. This is clearly not the correct solution as can be seen by

Figure 3: An example of a value sketch and the finished painting.

looking at the DKL table. In order to solve this problem a more in-depth explanation of the DKL color model is required.

5 Intensity Editing

We have developed a technique for manipulating the intensity values in a 3D computer generated image or photograph [2]. Artists often make *value sketches* of a scene before beginning the final painting (see Figure 3). This value sketch is based on the lighting of the scene, but is not simply a faithful copy of it. The artist uses the value sketch to experiment with the placement and tone values of large masses of light and dark areas. This results in increased coherency and focus in the final image, and also guides the viewer to the important elements of the image. Photographers and computer animators, on the other hand, do not have the luxury of simply changing the tone values of a rendering to create the image they want. Instead, they manipulate the lighting and the surface qualities of the scene objects to achieve the desired effect. A typical rendered scene in a high-quality animation film will have anywhere from ten to a hundred lights, placed in strategic locations. Similarly, photographers and film studios use spotlights and other special-purpose lighting to create better photographs.

We present an alternative method for adjusting a 3D computer generated image or photograph to achieve the desired tonal values. The user supplies a value sketch (which can be very rough) of the scene. Our system finds a mapping from the current shade values to new shade values that preserves the relative lighting and keeps as much contrast as possible. This mapping can be applied to the entire scene in order to attain the desired global tone values or to individual regions to attain the desired local variation.

The value sketch for a given 3D rendering or photograph can be created using a 2D paint program or by using traditional art media. We avoid the problem of aligning the value image with the original image by printing the original intensity image and making the value sketch on top of it. This is then scanned back in to the system. Another simple way to align the two images is to create a layer on top of the original intensity image in a 2D paint program and create the value sketch on that layer. Our system works by adjusting the tonal values of the actual rendered image so that they are closer to the tonal values of the value sketch without sacrificing contrast.

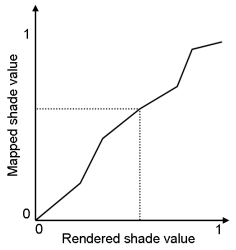


Figure 4: The value-map function.

5.1 Formal Problem Statement

Let I_R be some input intensity image and I_T be some target intensity image. Find a value mapping function $V : V(I_R) \longrightarrow I'_R$ such that:

1. $\forall$ pixels (x,y),
$$\sum_{x,y}(I'_{R_{x,y}} - I_{T_{x,y}})^2$$
is minimized. i.e. the absolute lighting in I'_R corresponds to I_T.

2. I'_R preserves the relative lighting that was present in I_R.

3. Contrast in I'_R is mazimized.
i.e the slope of V is maximized.

5.2 Representation of V

The value mapping function V must be monotonically increasing in order to preserve the relative lighting present in I_R. In addition $\forall$ input pixel values a, $0 \leq V(a) \leq 1$. This ensures that the resulting image I'_R can be displayed correctly. The default mapping function is simply the identity function.

To represent the value-map function we use a piece-wise linear function (see Figure 4). We could use a higher-order function, such as a quadratic spline function, but have found the linear function to be sufficient. Although the mapping is not smooth, if the divisions are small (0.1 or less) the effects are not noticeable.

5.3 Approach

Figure 5 show a flow diagram of our intensity editing technique. The basic idea is as follows:

1. The user supplies the original color image and a grayscale target intensity image (value sketch).

2. The original intensity image is obtained by converting the original color image to grayscale.

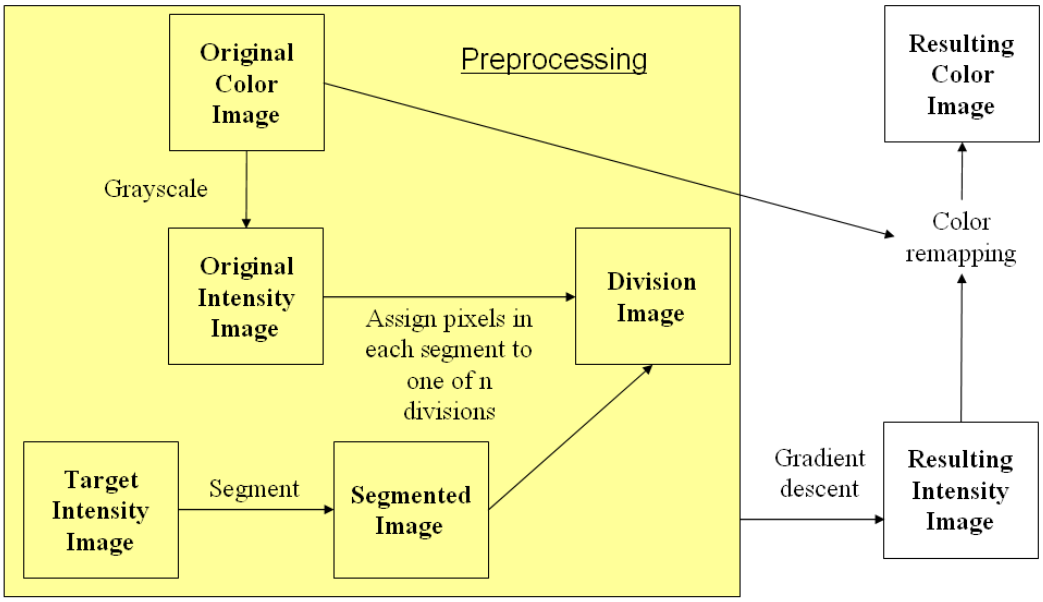


Figure 5: Flow Diagram.

3. The target intensity image is segmented.
4. The pixels within each segment are uniformly assigned to one of n (user input) divisions. These n divisions represent the n 'pieces' of the value mapping function V.
5. Gradient descent is used to determine a value mapping function V for each segment.
6. Applying the resulting value mapping functions to all pixels within their respective segments produces the result intensity image.
7. Finally the color is remapped. This produces the resulting color image.

The target intensity image is segmented using Comaniciu and Meer's color segmentation technique [3].

5.4 Gradient Descent

There is an initialization step in the gradient descent routine. We first determine where the x-values for the divisions of the segment lie. We then set the y-values equal to the x-values. In this way our starting function is the identity. During the gradient descent process, The y-values are adjusted until they minimize the weighted sum of three error functions:

$$Err_{Tot} = (Weight_A * Err_A) + (Weight_C * Err_C) + (Weight_M * Err_M)$$

Where:

- Err_{Tot} is the total error
- Err_A is the absolute error
- Err_C is the contrast error
- Err_M is the monotonically increasing error.
- $Weight_A$ is the weight for absolute error.
- $Weight_C$ is the weight for contrast error.
- $Weight_M$ is the weight for monotonically increasing test error.
  These weights should always sum to 1.0. A detailed explanation of the error functions is presented below.

The absolute error is calculated as follows:

$$Err_A = \frac{\sum_{x,y} \left| (I'_{R_{x,y}} - I_{T_{x,y}}) \right|}{num\_pixels}$$

Where

- I'_R is the image generated from some value mapping function.
- I_T is the target intensity image.
- (x,y) are all pixels in the segment being processed.

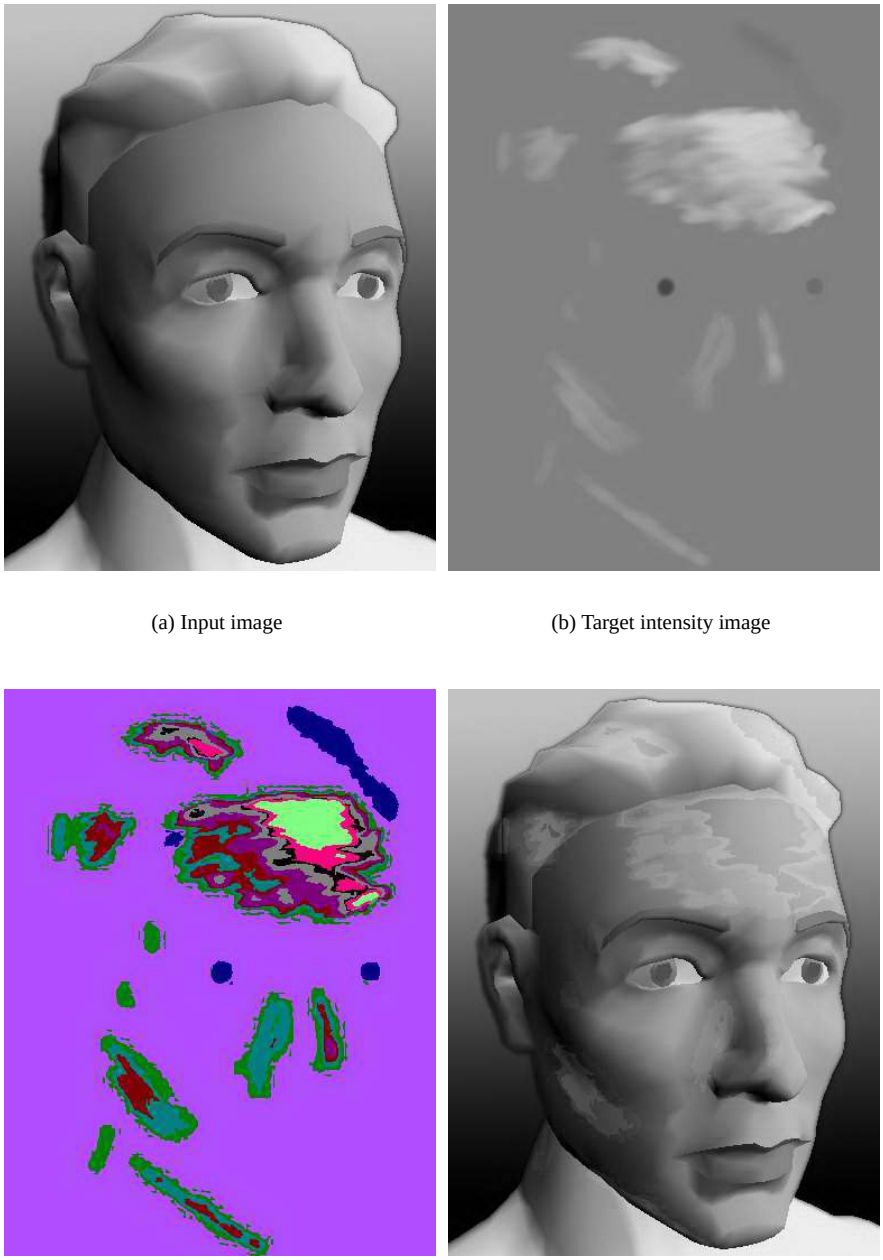
The contrast error is calculated as follows:

$$Err_C = \frac{\sum_i |(CurrYvalue[i] - MaxContrastYvalue[i])|}{num\_Yvals}$$

Where

- CurrYvalue[i] is the current y-value.
- MaxContrastYvalue[i] is the y-value that will maximize contrast.

The monotonically increasing error is calculated as follows:
$Err_M = 0.0$ if $CurrYvalue[i+1] - CurrYvalue[i] > 0$ $Err_M = 1.0$ otherwise.



(a) Input image

(b) Target intensity image

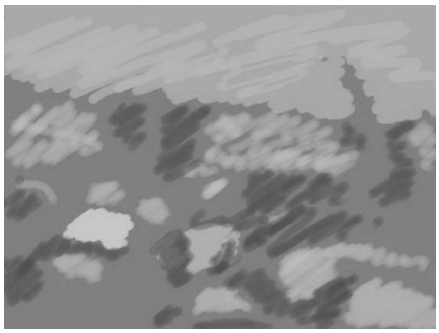
(c) Segmented image (Colored for illustration purposes)

(d) Result

Figure 6: Results for a typical rendered image.

(a) Input color image

(b) Original intensity image



(c) Target intensity image

(d) Resulting intensity image

(e) Resulting color image

Figure 7: Results for a typical photograph.

5.5 Results

Figure 6 and Figure 7 show some results obtained using our intensity editing technique. For the examples, the same weights were used for each segment. This causes the results for some segments to be closer to the target values than others. Evidence of this can be seen in Figure 6 on the forehead where the lightest target area is not the lightest area in the result. We are currently trying to develop techniques that analyze the image and automatically generate weights for each segment.

6 Current and Future Work

We are currently trying to incorporate the DKL color model with our intensity editing technique. We are also working on gaining a better understanding of the DKL color space boundary. We plan to incorporate the rod response into our system and develop perception based contrast and detail editing techniques.

References

[1] P. Lennie A. M. Derrington, J. Krauskopf. Chromatic mechanisms in lateral geniculate nucleus of macaque. *Journal of Physiology*, 357:241 –265, 1984.

[2] R. Bailey and C. Grimm. Using value images to adjust intensity in 3d renderings and photographs. 31st International Conference on Computer Graphics and Interactive Techniques, 2004. Presented at Poster Session.

[3] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 750. IEEE Computer Society, 1997.

[4] Krauskopf J. Lennie, P. and G. Sclar. Chromatic mechanisms in striate cortex of macaque. *Journal of Neuroscience*, 10:649 – 669, 1990.